ASTC/MGA 2001 Paper Number M097

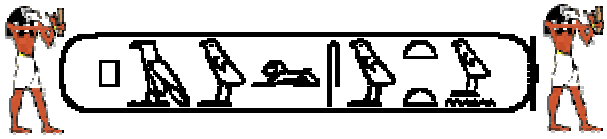
Interoperability Strategy Rosetta Stone

Paul W. Sutton
Interoperability Manager
Warfare Analysis, Modeling, and Simulation (PMW 153)
Space and Naval Warfare Systems Command (SPAWAR)
4301 Pacific Highway
San Diego, CA 92110-3127
(619) 553-9632
psutton@spawar.navy.mil

Keywords:
C4ISR System, DII COE, HLA, Interoperability, JTA, Model

ABSTRACT: Interoperability strategies are prescribed and implied by DoD, Joint, and Service interoperability policy documents, especially master plans, but the technical alternatives for specifying a distributed computing architecture are not described in terms of network and application cost/performance trade-offs. What are the technical alternatives? What are their cost/performance trade-offs? Do the alternatives reflect the most recent and most promising technology advancements? Are they flexible enough to easily incorporate and integrate future technology advances? Are they likely to result in significant improvements to real, physical, end-to-end interoperability? This paper provides a "Rosetta Stone," in the form of two hyper cubes, to help C4ISR system and simulation program managers, developers, and users, summarize, sort out, compare, contrast, and understand the cost/performance trade-off implications of selecting a specific technical interoperability strategy. One hypercube depicts three network performance trade-offs, scalability, reliability, and delay (latency), and the other hypercube shows application performance trade-offs, runtime, complexity, and development time/cost, associated with the adoption of each technical interoperability strategy. Together, they enable a C4ISR system or simulation program manager or developer to carefully weigh the performance trade-offs associated with selecting a particular technical interoperability strategy before making a final decision.

1. INTRODUCTION



When French Engineer M. Boussard unearthed the Rosetta stone in 1799, it ultimately revealed the secret of translating Egyptian hieroglyphics, something that had eluded man for almost two thousand years. (Brewer 2000)

> *"The hardest thing in the world to understand is income tax." -- Albert Einstein*

Interoperability strategy, though not as difficult to understand as hieroglyphics or income tax, has remained extremely difficult to comprehend. This article provides a *Rosetta stone* to C4ISR (Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance) system and simulation program managers, developers, and users that will enable them to unravel the interoperability strategy puzzle.

2. INTEROPERABILITY STRATEGY

The purpose of this paper is to help C4ISR system and simulation program managers, developers, and users, summarize, sort out, compare, contrast, and understand the cost/performance trade-off implications of selecting a specific technical interoperability strategy. *Interoperability* is severally defined here as:

> *"The ability of the systems, units, or forces to provide services to and accept services from other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together. The conditions achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users."* (CJCS 1995)

> *"The ability of a model or simulation to provide services to and accept services from other models and simulations, and to use the services so exchanged to enable them to operate effectively together."* (DoD 1994)

S*trategy* is used here to mean a set of goals or objectives, and the means of achieving them. It is an approach or method of getting something done. (Comerford and Callaghan 1999)

3. HYPERCUBES – FOR VISUALIZING COST/PERFORMANCE TRADEOFFS

The approach taken to analyzing interoperability strategies is to plot each strategy with two sets of performance attributes or dimensions, in two separate hyper cubes, with one hypercube representing network performance dimensions, and the other hypercube representing application performance dimensions, as shown in Figures 3.1 and 3.2. Various interoperability strategies can then be plotted as points in the three-dimensional space of each hypercube, with the origin of each hypercube representing the reference point, *architecture compliance (HLA and DII COE)*, for all the other strategies. Each axis or dimension in each hypercube is a nominal scale, with each plotted point representing nominal values (more than, less than, the same as) normalized to the origin or reference point. The performance dimensions in each hypercube are assumed to be orthogonal to one another to permit each interoperability strategy to be plotted in three-dimensional space, even though they may not always be completely independent of each other all the time. In this manner, the cost-performance tradeoffs of each interoperability strategy can then be visualized, compared, contrasted, and more easily understood.

3.1 Network Performance Dimensions Hypercube

As can be seen in Figure 3.1, network performance dimensions include scalability, delay, and reliability, where:

Scalability is defined as:

> "… the ability of a computer application or product (hardware or software) to continue to function well as it (or its context) is changed in size or volume in order to meet a user need. Typically, the rescaling is to a larger size or volume. The rescaling can be of the product itself (for example, a line of computer systems of different sizes in terms of storage, RAM, and so forth) or in the scalable object's movement to a new context (for example, a new operating system). It is the ability not only to function well in the rescaled situation, but to actually take full advantage of it." (Thing 2001)

Delay is used as "1) In a network, latency, a synonym for delay, is an expression of how much time it takes for a packet of data to get from one designated point to another. In some usages (for example, AT&T), latency is measured by sending a packet that is returned to the sender and the round-trip time is considered the latency. 2) In a computer system, latency is often used to mean any delay or waiting that increases real or perceived response time beyond the response time desired." (Thing 2001) and …

Reliability is taken to mean,

"An attribute of any system that consistently produces the same results, preferably meeting or exceeding its specifications. The term may be qualified, e.g., software reliability, reliable communication." (Howe 1997)

Similarly, *reliable communication* is, "Communication where messages are guaranteed to reach their destination complete and uncorrupted and in the order they were sent." (Howe 1997)

3.2 Application Performance Dimensions Hypercube

Application performance dimensions are similarly depicted in Figure 3.2. They include development time/cost, complexity, and runtime, where:

The operational definition of application *development time/cost* used here is "the total man-hours, calendar time, and monetary cost required to design, develop, test, document, market, and package a fully operational version of the application."

Application *complexity* can be defined in terms of its algorithms, "The level in difficulty in solving mathematically posed problems as measured by the time, number of steps or arithmetic operations, or memory space required (called time complexity, computational complexity, and space complexity, respectively). The interesting aspect is usually how complexity scales with the size of the input (the "scalability"), where the size of the input is described by some number N. Thus an algorithm may have computational complexity O(N^2) (of the order of the square of the size of the input), in which case if the input doubles in size, the computation will take four times as many steps." (Howe 1997)

Application *complexity* can also be defined in terms of the number of source code fragments (modules) and the number of control flow paths that connect them in a flowgraph of the application. A well-known metric, *Cyclomatic Complexity*, *V*=E-N+2, was devised to

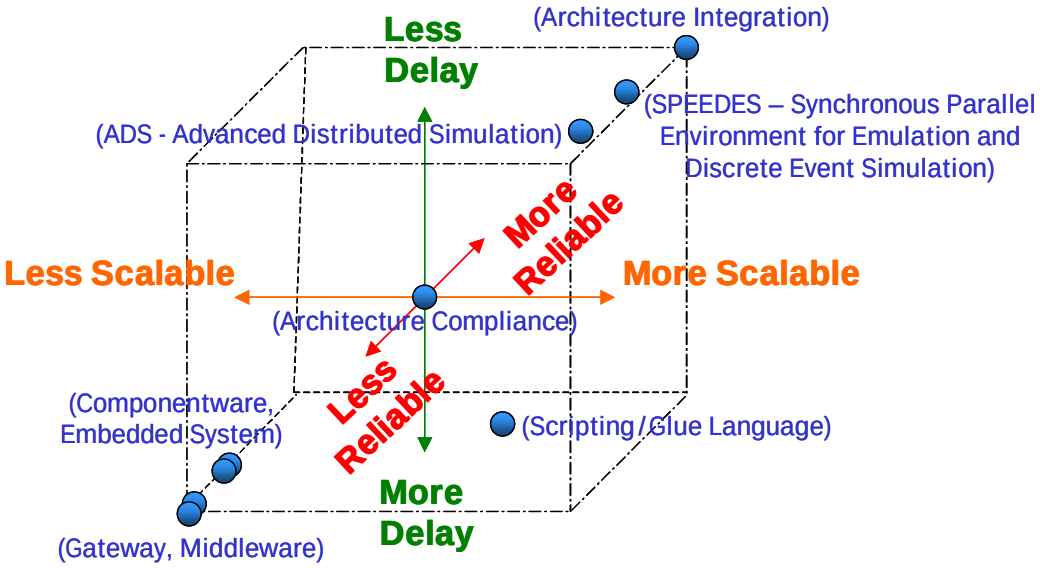


Figure 3.1 Interoperability Strategies' Network Dimensions

Hypercube Octant Network Dimension Values	Interoperability Strategies
Hypercube Origin, Center, Reference Point	Architecture Compliance: DII COE and HLA
Less Scalable, Less Reliable, More Delay	Componentware, Embedded System, Gateway, Middleware
More Scalable, Less Reliable, More Delay	Scripting / Glue Language
More Scalable, More Reliable, Less Delay	ADS – Advanced Distributed Simulation, SPEEDES – Synchronous Parallel Environment for Emulation and Discrete Event Simulation, Architecture Integration

Table 3.1 Interoperability Strategies' Network Dimensions

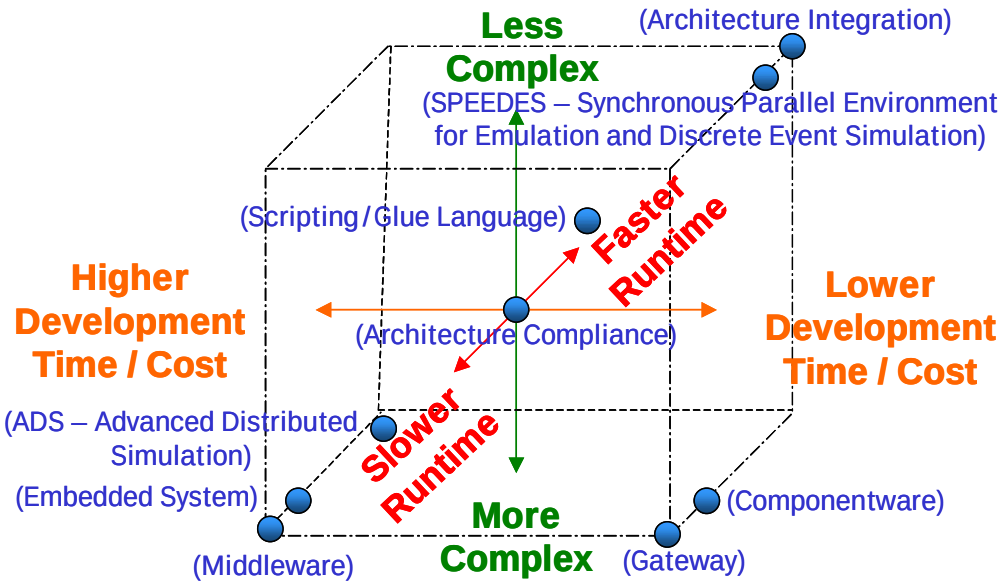


Figure 3.2 Interoperability Strategies' Application Dimensions

Hypercube Octant Network Dimension Values	Interoperability Strategies
Hypercube Origin, Center, Reference Point	Architecture Compliance: DII COE and HLA
Higher Development Time/Cost, Slower Runtime, More Complex	Embedded System, Middleware
Higher Development Time/Cost, Faster Runtime, More Complex	ADS – Advanced Distributed Simulation
Lower Development Time/Cost, Slower Runtime, More Complex	Componentware, Gateway
Lower Development Time/Cost, Slower Runtime, Less Complex	Scripting/Glue Language
Lower Development Time/Cost, Faster Runtime, Less Complex	SPEEDES – Synchronous Parallel Environment for Emulation and Discrete Event Simulation, Architecture Integration

Table 3.2 Interoperability Strategies' Application Dimensions

quantify this type of complexity. (McCabe 1976) Significantly, there is a correlation between complexity and the number of faults in an application. (Yacoub, et al. 1999) and …

Runtime is defined as, "the period of time during which a program is being executed, as opposed to compile time or load time." (Howe 1997)

4. INTEROPERABILITY STRATEGIES – ANALYSIS AND DISCUSSION

Interoperability strategies are plotted in Figures 3.1 and 3.2, and are listed in Tables 3.1 and 3.2. The rationale for plotting each strategy is analyzed, discussed, and explained in the following sections.

4.1 Architecture Compliance

Two distributed computing architectures are mandated by the Joint Technical Architecture (JTA) for military applications, DII COE (Defense Information Infrastructure Common Operating Environment) for C4ISR system applications, and the HLA (High Level Architecture) for M&S (Modeling and Simulation) applications. (DISA 1997a) (USD(A&T) 1996) Since all military applications must comply with these architectures, architecture compliance is used here as the reference point for normalizing cost and performance attributes of all the other interoperability strategies.

4.1.1 DII COE (Defense Information Infrastructure Common Operating Environment).

DII COE provides a standardized operating environment for command and control applications. It includes application program interfaces (APIs) for different types of applications such as Service Command and Control (C2), common support applications (e.g., message processing), infrastructure services (e.g., communications), a kernel (e.g., Motif), and a shared data environment (SHADE) that includes various databases such as intelligence. These interfaces and services are implemented as concentric layers that reside on top of the host computer's operating system. (DISA 1997b)

4.1.1.1 DII COE Advantages and Disadvantages. DII COE's principle advantages include:

- Segmented C4ISR applications behave in a predictable, familiar, consistent, uniform manner.
- Compliant applications are generally compatible with one another.
- DII COE standards and specifications are closely controlled by DISA with frequent reviews and widespread participation by the Services and other users.

DII COE also has some disadvantages:

- DII COE compliance at Level 5 does not ensure that C4ISR applications will necessarily be interoperable with one another. (Sutton 1999)
- DII COE does not provide all the interfaces or services required by simulations, e.g., time management. As a result, simulations must run on a separate distributed computing architecture such as HLA. This prevents C4ISR applications from being inherently interoperable with simulations.
- DII COE is not a true open-systems standard in the sense that other distributed computing architectures are, such as CORBA (Common Object Request Broker Architecture).

4.1.2 HLA (High Level Architecture).

"HLA is the technical architecture for DoD Simulations. (It provides) major functional elements, interfaces, design rules, pertaining to all DoD simulation applications, and providing a common framework within which specific system architectures can be defined." (AEgis 1998)

4.1.2.1 HLA Advantages and Disadvantages. An excellent comparison of three distributed computing architectures, HLA, CORBA, and RMI (Remote Method Invocation), which clearly explains their relative advantages and disadvantages, is summarized in Table 4.1.2.1. (Buss and Jackson 1998)

Additional objections to HLA include (Davis and Moeller 1999):

- HLA does not scale well. It scales as N^2, while other distributed computing architectures, such as CORBA, scale as N, because they provide direct communication between objects.
- It doesn't provide a hierarchical system-of-systems modeling framework for C4ISR systems.
- It isn't extensible or evolvable to new distributed computing environments such as on-line planning and control, and multiple-use design, execution, and training.
- It acts as a barrier to incorporating advancements in computing and networking technologies because it was designed to preserve and reuse existing legacy models and simulations.

ATTRIBUTE	HLA/RTI	CORBA	RMI
Types of Applications Supported	Legacy	Legacy	New
Programming Languages Supported	Ada, C++, Java, CORBA IDL	Ada, C, C++, Java, OO-COBOL, Smalltalk	Java
Direct Communications Between Objects Provided	No	Yes	Yes
Transfer of Object Ownership Provided	Yes	No	No
Time Management Services Provided	Yes	No	No
Security Management Services Provided	No	No	Yes
Network Communications Protocols Specified	Left to RTI vendor	IIOP	TCP/IP
Cross-Language Compatibility Overhead	RTI implementer	None	Not compatible
Meta-Language Provided	None	IDL	None

Table 4.1.2.1 Comparison of Distributed Computing Architecture Attributes

Other serious HLA problems have been identified (Nance 1999):

- HLA provides data exchange mechanisms such as data formats, but no metadata. As a result, data exchange can't evolve as technology changes.
- HLA suffers from time management problems. Dissimilar time management mechanisms, e.g., one federate time-stepped (e.g., EADSIM), and one federate event-stepped (e.g., NSS), introduce computational inefficiencies, such as slower runtime.
- Incompatible timing granularity (time scale differences) can cause serious faults in parallel federates. Conservative federates may experience slower runtimes or crashes, and optimistic federates have a higher chance of having time errors.

4.2 Gateway

A *gateway* is, "a device that connects two systems, especially if the systems use different protocols. For example, a gateway is needed to connect two independent local networks, or to connect a local network to a long-haul network." (Stallings 1994)

A *transport gateway* "moves data from one type of network to another, typically by embedding all of the information associated with one type of protocol into another." There are two approaches used. *Tunneling* involves passing traffic from one type of network through another type of network, e.g., one gateway inserts SNA data into TCP/IP packets and another gateway extracts the data from the packets. With *encapsulation*, a workstation embeds a foreign protocol into its local protocol and sends it to a gateway, which then extracts it and sends it to the appropriate network. Since gateways introduce overhead and delay into the network, they are plotted in the hypercube octants with more delay and slower runtime. (Breit 1999)

4.2.1 Gateway Advantages and Disadvantages.

The primary advantage of using a gateway is its ability to permit an application written for one language, system, and architecture, to interoperate with an application written for a different language, system, and architecture, without having to reprogram either application. The *gateway* interoperability strategy, therefore, is plotted in the hypercube octant with lower development time/cost. Legacy applications written for the DIS (Distributed Interactive Simulation) architecture, for example, can interoperate with HLA-applications without having to rewrite them to be compliant with HLA.

Unfortunately, gateway disadvantages are legion and they far outweigh its primary advantage. Significant gateway disadvantages include (Dodge 2000):

- A point-to-point gateway connection creates a single point of failure, which reduces system reliability.
- Gateways can become communications bottlenecks that reduce communications reliability. The Army Aviation and Missile Command (AMCOM), for example, experienced severe data anomalies, such as erroneous roll rates, altered data, and appearance field perturbations, in its HLA/DIS gateway. (Hall, et. al. 1998)
- Gateways introduce delays into a network, which can result in slower application runtime and network instability. AMCOM, for example, experienced network delays of up to 12 seconds during periods of peak DIS-HLA translation. They also experienced slower application runtimes and crashes caused by gateway instability and gateway disenfranchisement with HLA RTI (Run Time Infrastructure). (Hall, et. al. 1998)
- Gateways scale poorly (N^2), which makes system implementation and maintenance more complex.

Gateways, for the reasons noted above, are plotted in the *Less Scalable, More Delay, Less Reliable, Slower Runtime, More Complex, Lower Development Time/Cost* octants of Figures 3.1 and 3.2. It is also interesting to note that networking technology advances may eventually eliminate the need for gateways. High-speed LAN(Local Area Network)s that run Internet Protocol will plug into WAN(Wide Area Network)s via Sonet technology on carrier backbones, or by plugging directly into the WAN's fiber-optic backbone. (Adhikari 1998)

4.3 Middleware

Middleware is "a class of software whose purpose is to simplify the complex problem of developing and using applications on different platforms of computers, connected over different types of networks. In other words, Middleware is a layer of software that supports multiple communication protocols, multiple programming languages, and can be executed on different computer platforms. It fits between the application program and the network interface, and is in effect a set of Application Program Interfaces (APIs) that software programmers can use to avoid concerns about the underlying network and operating system software." (Whiting 1994)

4.3.1 Middleware Advantages and Disadvantages.

As pointed out in its definition, the main advantage of middleware is that it allows application programs to be written in different programming languages to operate on different kinds of computer hardware, operating systems, and networks, without having to rewrite the application program for each different environment.

Unfortunately, middleware advantages are offset by a number of disadvantages:

- Scalability is an important problem. "Most middleware has been designed with procedural protocols in mind, but many systems evolve to use various nonprocedural ones: Multicast-based event frameworks, remote-execution frameworks, and mobile agent frameworks all exist in one form or another, but the most common versions are all uncomfortably grafted on the same old stuff." (Milojicic 1999)
- Delay is another significant problem, "… conventional ORB-based systems incur significant throughput and latency overhead." (Campbell, et al. 1999)
- Reliability is also considered problematic. "In many cases, OMG found, code that runs above the operating system is more error prone than the commercial RTOSes (real-time operating systems), which have been wrung out well during years of usage." (Costlow 1998)
- Complexity and application development time/cost present even greater difficulties. Middleware adaptations to new types of applications, such as multimedia, real-time, and mobility, have spawned adaptations and evolutions such as *Minimal CORBA, Real-time CORBA, and interceptors*, but

"… in general there is no principled approach to these adaptations and evolutions. They are carried out in ad-hoc ways, yielding application programming difficulties and inordinate system complexity." (Eliassen, et al.1999)

- Application runtime is also problematic. "The current generation of middleware has a deserved reputation for running slowly. In an ORB-based middleware system, developers simply model the legacy component using the same IDL they use for creating new objects, then write *wrapper* code that translates between the standardized bus and the legacy interfaces." (Campbel, et al. 1999)

Considering the numerous examples above, middleware is plotted in the *Less Scalable, More Delay, Less Reliable, Higher Development Time/Cost, More Complex, and Slower Runtime* octants of the hyper cubes.

4.4 Componentware

Componentware or *Component Software* is "…application development … using larger building blocks than lines of code. DARPA used to call this mega-programming." It promises " … rapid application assembly from components -- Leggo-like reuse to build large systems from known components." "Components themselves do not have to be tested and retested. It may be possible to derive properties of configurations of components from the properties of the component parts and the glue holding components together. Because all of the interfaces between components are standardized, it is possible to mix components from different manufacturers in a single system. Similarly, the goal of component software is to standardize the interfaces between software components so that they too can work together seamlessly." (*Componentware Glossary* 2001)

4.4.1 Componentware Advantages and Disadvantages.

Componentware's chief advantage is its promise of lower application development time and cost, but it may be awhile before that promise is realized. *DCOM* (Distributed Component Object Model) *APIs* (Application Programming Interfaces), for example, "are not structured and presented in an intuitive way for building distributed client/server applications. As a result, … the DCOM architecture often involves non-intuitive programming hacks to provide the functionality required in a client/server environment." (Wang, et al. 1997)

Componentware's numerous disadvantages include:

- Scalability is a problem because "… deployment of … (distributed, loosely-coupled, heterogeneous, asynchronous event-driven) systems at the scale of the Internet imposes new challenges that are not met by existing technology. In particular, the technology to support an event-based architectural style is well developed for local-area networks, … but not for wide-area networks. One of these systems, *Yeast*, … is a general-purpose platform for building distributed applications in an event-based architectural style, and it supports event-based interaction quite naturally within local-area networks. However, its centralized-server architecture limits its scalability to wide-area networks." (Rosenblum, et al. 1998)
- Componentware can also introduce delays into a network – "There are a number of challenging issues that need to be addressed for this approach: communication delays have to be taken account, as well as the overhead to locate CORBA objects because of CORBA location transparency." (Le and Chakravarthy 1998)
- Reliability of componentware is also troubling. "The OS (Operating System) is oblivious to the component abstraction and cannot effectively provide a service tailored to individual components. In a component-based application, a file opened by one component might inadvertently be manipulated or closed by another. Further, lack of isolation tends to result in one component's bugs crashing another ..." (Mendelsohn 1997)
- Componentware seems to be inordinately complex at its present level of maturity. "The introduction of composable simulation, while providing improved flexibility in simulations, can actually increase the level of complexity in creating a particular target simulation." (Aronson 2000) "Classes/objects implemented in one programming language cannot interoperate with those implanted in other languages. In some object-oriented languages even the same compiler version has to be used so that objects become interoperable." (Pree 1997)
- Runtime also seems problematic, "Although frequent procedure calls and message sends are important structuring techniques in object-oriented languages, they can also severely degrade application run-time performance." (Grove 1998)

One can conclude, therefore, that *componentware* should be plotted in the *Less Scalable, More Delay, Less Reliable, Lower Development Time/Cost, More*

Complex, and Slower Runtime octants of the hyper cubes

4.5 Embedded System

An *embedded system* is "Hardware and software which forms a component of some larger system ... Often it must provide real-time response." (Howe 1997) This definition is oriented toward microprocessor implementations. In the broader sense, this definition is used here to include *embedded software*, *embedded simulation*, and *embedded training* – it might only include software.

4.5.1 Embedded System Advantages and Disadvantages.

- Embedded systems do not generally scale well, "… RPC toolkits are well suited for conventional request/response-style applications running on low-speed networks. Until recently, however, the QoS specification and enforcement features of conventional DOC middleware and ORBs, as well as their efficiency, predictability, and scalability, have not been suitable for applications with hard real-time requirements." (Schmidt 1999)
- Embedded system delay and runtime are problematic. "Conventional ORBs often incur significant throughput and latency overhead. This overhead stems from excessive data copying, non-optimized presentation layer conversations, internal message buffering strategies that produce non-uniform behavior for different message sizes, inefficient demultiplexing algorithms, long chains of intra-ORB virtual method calls, and lack of integration with underlying OS and network QoS mechanisms." (Schmidt 1999)
- The reliability of embedded systems is also cause for concern. "… CORBA, DCOM, and RMI do not require an ORB to notify clients when transport layer flow control occurs. Therefore, it is hard to write portable code and efficient real-time applications that will not block when ORB end-system and network resources are temporarily unavailable. Likewise, conventional DOC ORBs do not propagate exceptions stemming from missed deadlines from servers to clients, which makes it hard to write applications that behave predictably when congestion in the communication infrastructure or end-systems causes deadlines to be missed." (Schmidt 1999)
- Similarly, application development time, cost, and complexity are greater than they are for other interoperability strategies. Whenever one system is designed and built to reside within another, dissimilar system, the developer must then account for new interfaces and interactions that are often novel, unplanned, and unexpected. "… real-time, embedded system software development has historically lagged mainstream software development methodologies. As a result, real-time embedded software systems are costly to evolve and maintain. Moreover, they are so specialized that they cannot adapt readily to meet new market opportunities or technology innovations." (Schmidt 1999)

There has been considerable research directed at making simulations interoperable with C4ISR systems, and *embedded systems* are one of the specific strategies being considered. Various approaches within this strategy include: (1) embedding the simulation application directly within the C4ISR application (a.k.a. *embedded simulation* or *embedded training*); (2) embedding HLA RTI within DII COE at the *common applications* level; (3) at the *common services* level; and (4) at the *kernel* level. Since both DII COE and HLA RTI are forms of middleware built on top of operating systems, and since both provide *data and object management services*, how will the embedded system reconcile how two different sets of common services are accessed and used? How will one application, service, or operating system avoid becoming disenfranchised with the other?

Considering the above advantages and disadvantages, it seems reasonable to plot *embedded systems* in the *Less Scalable, More Delay, Less Reliable, Higher Development Time/Cost, More Complex, and Slower Runtime* octants of the hyper cubes.

4.6 Scripting/Glue Language

Scripting or *Glue* languages "assume that a collection of useful components already exist in other (procedural) languages. They are not intended for writing applications from scratch but rather for combining components." Examples of scripting languages include Perl, Python, Rexx, Tcl, Visual Basic, and Unix shells. (Ousterhout 1998)

4.6.1 Scripting/Glue Language Advantages and Disadvantages.

Advantages include (Ousterhout 1998):

- Scripting languages provide better scalability because, "A type-less language makes it much easier to hook together components. There are no a priori restrictions on how things can be used."
- Scripting language lowers application development time and cost – "In every case, the scripting version required less code and development time than the system programming version."

- Application complexity is also reduced because scripting languages are type-less. New objects can be used with existing interfaces without having to write conversion code to translate between different types of objects.

Unfortunately there's still a downside:

- Scripting languages provide slower runtimes and longer delays than procedural languages because they use interpreters instead of compilers.
- The type-less nature of scripting languages is also a potential source of unreliability because gluing applications don't check for errors. Instead, that task is left to the components built with systems programming languages.

As indicated above, *Scripting/Glue Language* is plotted in the *More Scalable, Less Reliable, More Delay, Lower Development Time/Cost, Slower Runtime,* and *Less Complex* octants of the hyper cubes.

4.7 Advanced Distributed Simulation (ADS)

Advanced Distributed Simulation (ADS) has been defined as, "any application or architecture which employs the characteristics of distribution and networking in a way which permits a number of nodes, entities, or devices to interact with each other for some common or shared purpose …" (Murphy and Roane 1999) It was implemented in DARPA's Synthetic Theater of War (STOW) Advanced Concept Technology Demonstration (ACTD), which culminated in the largest entity level simulation ever used for a distributed training exercise.

4.7.1 ADS Advantages and Disadvantages.

ADS' advantages and disadvantages include (Cole, et al. 1998):

- Improved scalability through bi-level multicast communications (IP [Internet Protocol] multicast LAN-to-LAN service over less dynamic ATM (Asynchronous Transfer Mode) point-to-multipoint virtual circuit wide area service).
- Lower network delay and faster application runtimes due to low-latency, high speed ATM communications service, with assured low latency and guaranteed bandwidth through QoS (Quality of Service) reservation.
- Decreased reliability of unicast traffic. Multicast simulation data were signaled as VBR (Variable Bit Rate) point-to-multipoint SVCs (Switched Virtual Circuits), while unicast traffic was sent over UBR (Unspecified Bit Rate) SVCs. VBR traffic had priority over UBR traffic, which was dropped first if network congestion was encountered.
- Application complexity and development time/cost were somewhat higher with ADS than with simple architecture compliance (HLA/RTI on LAN) because special network communications equipment, such as the QCBMR (QoS-Capable, Bi-level, Multicast Router), had to be developed to support some of the new, advanced networking techniques.

For these reasons, ADS is portrayed as a *More Scalable, Less Delay, Less Reliable, Higher Development Time/Cost, More Complex,* and *Faster Runtime* interoperability strategy in the hyper cubes.

4.8 SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation)

SPEEDES is a distributed simulation architecture that was specifically designed for parallel discrete-event simulation (PDES). It was developed by JPL for NASA in 1990 and is licensed by NASA. It includes application programming interfaces, management services, a modeling framework, time management, an event processing engine, a communications library and interfaces. The Office of Naval Research selected SPEEDES in 1996 as the most promising simulation execution framework for simulation development. (Wallace 2000)

4.8.1 SPEEDES Advantages and Disadvantages.

SPEEDES' advantages and disadvantages include:

- SPEEDES scales very well. For example, total event and message memory consumption during application execution increased only 10 percent as the number of processors was increased from 32 to 96 nodes. (Steinman, et al. 1999)
- Network delay is also less than that experienced with other distributed computing architectures. There has been less delay for shared memory implementations (e.g., 11.3 usec) than TCP/IP implementations (2 msec). Latency as a function of message size remains low and constant for small messages, and then increases exponentially when the message begins to exceed the size of TCP/IP packets. (Van Iwaarden 1999)
- SPEEDES is considered more reliable (Wallace 2000) than other distributed computing architectures by its proponents, but others warn that reliability problems may still be encountered when "synchronizing a parallel discrete-event simula-

tion: simulation code that runs correctly on a serial machine may, when run in parallel, fail catastrophically." (Nicol and Liu 1997)

- SPEEDES application runtimes are also superior to those of other distributed computing architecture – which is what one would expect with parallel processors. Recent tests demonstrated "nearly perfect speedup when going from 32 to 96 processors." (Steinman, et al. 1999)
- Application complexity is similarly reduced by the use of SPEEDES – "This mapping, or API translation, is implemented in a compatibility library, and allows the infrastructure model software to be linked to the SPEEDES libraries for runtime execution as a virtual single, composite simulation." (Walace, et al. 2000)
- SPEEDES also seems to contribute to lower application development time and cost. JSIMS Maritime, for example, achieved an average productivity of 640 SLOC/person-month, which compared quite favorably with an industry average of 200 SLOC/person-month. (Wallace, et al. 2000)
- Recent experience with SPEEDES, therefore, suggests that it should be plotted as a *More Scalable, Less Delay, More Reliable, Lower Development Time/Cost, Less Complex*, and *Faster Runtime* interoperability strategy in the hyper cubes.

4.9 Architecture Integration

Architecture integration is used here to mean an entirely new, holistic distributed computing architecture that combines the advantages of each interoperability strategy while rejecting its disadvantages. It means building a single adaptable, evolvable, open architecture for both C4ISR applications and simulations that will optimize both network and application performance in a heterogeneous environment.

4.9.1 Architecture Integration Advantages and Disadvantages.

The *architecture integration* interoperability strategy is plotted in the optimum position of each hypercube because it represents the ideal strategy, with all of the strengths and none of the weaknesses of each individual strategy. Some may view architecture integration as an overly ambitious and idealistic *holy grail*, while others view it as a perfectly logical conclusion of previous, like-minded research efforts.

5. CONCLUSIONS

The following conclusions may be drawn from the above analysis, discussions, and explanations:

- Network and application performance attributes, development time, and development cost can be evaluated and plotted in hyper cubes that permit program managers, system developers, and users to visualize, compare, and contrast their relative tradeoffs. In this paper plotting points representing contending interoperability strategies on nominal scales of orthogonal network and application performance and cost dimensions did this.
- The hypercube approach to visualizing and understanding cost/performance tradeoffs of interoperability strategies requires some simplifying assumptions that can rightfully be challenged. Performance dimensions are portrayed as being orthogonal to one another, but it is obvious that they are not entirely independent of each other. Network delays, for example, are related to communications reliability. Application complexity is related to development time and cost, and so forth. The author understands this, but believes that the violation of strict independence is far outweighed by our ability to plot and visualize each strategy in the octant of the hypercube that represents the appropriate combination of nominal values. Also, each strategy is not mutually exclusive to the all the others. Some strategies may be combined with others in a particular distributed computing architecture. Gateways, for example, are often used with HLA. The definitions of some strategies overlap. HLA, for example, is considered to be middleware. Nonetheless, the author believes that the usefulness of hyper cubes overshadows these simplifying assumptions.
- Figure 3.1 suggests that gateways and middleware provide the least relative advantage in terms of network performance tradeoffs, while ADS and SPEEDES provide the most. Scripting/Glue languages are only slightly better than gateways and middleware since they scale better and are useful for integration.
- Figure 3.2 indicates that middleware and embedded systems provide the least relative advantage in terms of application performance tradeoffs, while SPEEDES provides the most. ADS is the next best choice after SPEEDES because it provides faster application runtimes in long-haul environments through multicasting and QoS. Gateways and componentware are only slightly better than middleware and embedded systems -- because gateways eliminate the need to reprogram legacy applications to make them architecture compliant,

while componentware permits new applications to be built from reused components.

- Figures 3.1 and 3.2 both imply that SPEEDES is probably the best overall interoperability strategy in terms of tradeoffs. It seems to be closer to the optimum strategy, architecture integration, than any other strategy.
- Architecture integration is the optimum interoperability strategy in the author's view, because it would lead to a single, integrated, open systems, distributed computing architecture that could be used for both C4ISR and simulation applications. This would result in a single networking environment that would be vastly better and cheaper than the separate environments we have today. Unfortunately, political realities will probably prevent this from ever happening. Others also feel that too many domain-specific compromises would prevent us from ever achieving *universal* interoperability.

6. RECOMMENDATIONS

The following recommendations are made to improve interoperability:

- Future research should be directed toward benchmarking the performance and cost attributes of interoperability strategies – so that actual measurements of performance and cost can be plotted on interval scales (instead of the more subjective nominal scales). This research should be conducted in a controlled environment that facilitates repeatable experiments that yield consistent, comparable data.
- The DoD should combine and leverage its research efforts and resources to design and build a single distributed computing architecture that can be used for both C4ISR and simulation applications. *Architecture Integration* should be employed as the interoperability strategy for achieving this goal because it leverages the experience and resources previously used to build other contending architectures. This is the long-term solution.
- In the short-term, the most promising interoperability strategy that has demonstrated the best performance/cost tradeoffs should be selected for adoption and improvement, while less promising strategies should be discarded. SPEEDES should be examined more closely to determine whether it should be adopted as the best approach to achieving interoperability.

7. ACKNOWLEDGEMENTS

The author wishes to acknowledge the contributions of CAPT Joe Celano, Program Manager, Warfare Analysis, Modeling, and Simulation, Phillip Hornick, Deputy Program Manager, and Candace Conwell, Technical Director. This work would not have been possible without their keen insight and special efforts.

8. REFERENCES

Adhikari, R. 1998. "Search for End-to-end Reliability," *Information Week* (March 9), Issue 672, 10-14. Manhasset.

AEgis. 1998. "DoD HLA Process and Policy," *Comprehensive HLA Introduction* (21 April). Menlo Park, California.

Aronson, J. 2000. "Benefits and Pitfalls of Composable Simulation," in *Proceedings of the Spring Simulation Interoperability Workshop* (March). Orlando, Florida.

Breit, L. 1999. "Network gateways," *HP Chronicle* (Jan), Vol. 16, No. 2, 9-10. Austin, Texas.

Brewer, E.C. 2000. "Rosetta Stone," in *Dictionary of Phrase and Fable* (May). Bartleby.com, Philadelphia. http://www.bartleby.com/81/14522.html

Buss, A.H. and Jackson, L. 1998. "Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI," in *Proceedings of the 1998 Winter Simulation Conference*, Eds. D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, 819-825. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Campbell, A.T., Coulson, G., and Kounavis, M.E. 1999. "Managing Complexity: Middleware Explained," *IT Professional* (Sept-Oct), Vol. 1, No. 5, 22-28.

Chairman of the Joint Chiefs of Staff (CJCS) 1995. *CJCS Instruction 6212.01A, Compatibility, Interoperability, and Integration of Command, Control, Communications, Computers, and Intelligence Systems* (30 June), Enclosure A, p. A-3. Washington,D.C.

Cole Jr., R., Root, B., O'Ferrall, L., and Tarr, J. 1998. "STOW Network Technologies and Operatonal Lessons Learned," 98S-SIW-103, in *Proceedings of*

the Spring Simulation Interoperability Workshop (March). Orlando, Florida.

Comerford, D.W. and Callaghan, D.W. 1999. *Strategic Management: Text, Tools and Cases for Business Policy*, 2nd ed. Kent Publishing.

Componentware Glossary. 2001. http://www.objs.com/survey/ComponentwareGlossary.htm#GlossaryTerms

Costlow, T. 1998. "Real-time software world raises the reliability bar," *EETimes.com* (February 13) http://www.eet.com/news/98/994news/realtime.html

Davis, W.J. and Moeller, G.L. 1999. "The High Level Architecture: Is there a better way?" in *Proceedings of the 1999 Winter Simulation Conference*, Eds. P.A. Farrington, H.B. Nembhard, D.T. Surrock, and G.W. Evans, 1595-1601. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Defense Information Systems Agency (DISA). 1997a. *Joint Technical Architecture (JTA)*, Version 2.0 (1st draft) (31 October). Washington, D.C.

Defense Information Systems Agency (DISA). 1997b. "DII Perspective," *Defense Information Infrastructure (DII) Conference* (4 April). Washington, D.C.

Department of Defense (DoD). 1994. *Department of Defense Directive (DoDD) 5000.59, DoD Modeling and Simulation (M&S) Management* (4 January), Enclosure 2, 1. Washington, D.C.

Dodge, D.S. 2000. "Gateways: A Necessary Evil?" 00F-SIW-107, in *Proceedings of the Fall Simulation Interoperability Workshop* (September). Orlando, Florida.

Eliassen, F., Andersen, A., Blair, G.S., Costa, F., Coulson, G., Goebel, V., Hansen, O., Kristensen, T., Plagemann, T., Rafaelsen, H.O., Saikoski, K.B., and Yu, W. 1999. "Next Generation Middleware: Requirements, Architecture, and Prototypes," in *Proceedings of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, 60-65. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Grove, D. 1998. Ph.D Thesis: *Effective Interprocedural Optimization of Object-Oriented Languages*. http://www.cs.washington.edu/research/projects/cecil/www/Papers/grove-thesis.html

Hall, K.L., Bentley, T., Harless, W., Lee, G., Roose, K., and Sanders, B. 1998. "Performance and Interoperability Observations of the Gateway Approach to HLA Compliance for Legacy VR Simulations," 98F-SIW-131, in *Proceedings of the Fall Simulation Interoperability Workshop* (September). Orlando, Florida.

Howe, D., Ed. 1997. *The Free On-line Dictionary of Computing*. http://wombat.doc.ic.ac.uk/

Le, R. and Chakravarthy, S. 1998. "Support for Composite Events and Rules in Distributed Heterogeneous Environments," Paper No. 101, in *Proceedings of Workshop on Computational Software Architectures* (5-8 Jan). Monterey, California. http://www.objs.com/workshops/ws9801/program.html#II-3

McCabe, T. 1976. "A Complexity Metric," *IEEE Transactions on Software Engineering* (Dec), Vol. 2, No. 4, 308-320. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Mendelsohn, N. 1997. "Operating Systems for Component Software Environments," *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems* (5-6 May), 49-54. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Milojicic, D. 1999. "Middleware's role, today and tomorrow," *IEEE Concurrency* (April-June), Vol. 7, No. 2, 70-80. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Murphy, W.S. Jr. and Roane, M.L. 1999. "Application of the Analysis Federate in the Joint Advanced Distributed Simulation Joint Test Force Electronic Warefare Phase II Test," in *Proceedings of the 1999 Winter Simulation Conference* (5-8 Dec), Eds. P.A. Farrington, H.B. Nembhard, D.T. Surrock, and G.W. Evans, Vol. 2, 1109-1117.

Nance, R.E. 1999. "Distributed Simulation with Federated Models: Expectations, Realizations, and Limitations," in *Proceedings of the 1999 Winter Simulation Conference*, Eds. P.A. Farrington, H.B. Nembhard, D.T. Surrock, and G.W. Evans., 1026-1031. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Nicol, D.M. and Liu, X. 1997. "The Dark Side of Risk (What your mother never told you about Time Warp)," in *Proceedings of the 1997 Workshop on Parallel and Distributed Simulation* (10-13 June), 188-195. Austria.

Ousterhout, J.K. 1998. "Scripting: Higher Level Programming for the 21st Century," *Computer* (March), Vol. 31, No. 3, 23-30. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Pree, W. 1997. "Component-Based Software Development – A New Paradigm in Software Engineering?" in *Proceedings of the Asia Pacific Software Engineering Conference 1997 and the International Computer Science Conference 1997*, 523-524. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Rosenblum, D.S., Wolf, A.L., and Carzaniga, A. 1998. "Critical Considerations and Designs for Internet-Scale, Event-Based Compositional Architectures," Paper No. 033, in *Proceedings of Workshop on Computational Software Architectures* (5-8 Jan). Monterey, California. http://www.objs.com/workshops/ws9801/program.html #II-3

Schmidt, D.C. 1999. "Middleware Techniques and Optimizations for Real-time Embedded Systems," in *Proceedings of the 12th International Symposium on System Synthesis* (10-12 Nov), 12-16. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Stallings, W. 1994. *Data and Computer Communications*. Macmillan, Englewood Cliffs, New Jersey.

Steinman, J.S., Tran, T., Burckhardt, J., and Brutocao, J.S. 1999. "Logically Correct Data Distribution Management in SPEEDES," 99F-SIW-067, in *Proceedings of the Fall Simulation Interoperability Workshop* (September). Orlando, Florida.

Sutton, P. 1999. "Interoperability: A New Paradigm," in *Computational Intelligence for Modeling, Control & Automation: Neural Networks & Advanced Control Strategies*, Ed. M. Mohammadian, 351-361. IOS Press, Amsterdam.

Thing, Lowel, Ed. 2001. "Whatis.com," *TechTarget.com*. http://whatis.techtarget.com/WhatIs_Home_Page/0,4324,,00.html

Under Secretary of Defense (Acquisition & Technology) [USD(A&T)]. 1996. Memorandum, *DoD High Level Architecture (HLA) for Simulations* (10 September). USD(A&T), Washington, D.C.

Van Iwaarden, R., Steinman, J.S., and Blank, G. 1999. "A Combined Shared Memory and TCP/IP Implementation of the SPEEDES Communications Library," 99F-SIW-095, in *Proceedings of the Fall Simulation Interoperability Workshop* (September). Orlando, Florida.

Wang, Y., Damani, O.P., and Lee, W. 1997. "Reliability and Availability Issues in Distributed Component Object Model (DCOM)," in *Proceedings of the Fourth International Workshop on Community Networking* (11-12 Sept), 59-63. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey.

Wallace, J., Celano, J., and Peterson, L. 2000. "PANDA: An Approach to Simulatin Software Engineering, Development, and Integration," in *Proceedings of the 2000 Summer Computer Simulation Conference* (16-20 July), Ed. W.F. Waite, 353-360. Vancouver, British Columbia.

Whiting, R. 1994. "Getting on the Middleware Express," *Client/Server Today* (November), 70-75.

Yacoub, S.M., Ammar, H.H., and Robinson, T. 1999. "Dynamic Metrics for Object Oriented Design," in *Proceedings of the Sixth International Software Metrics Symposium* (4-6 Nov), 50-61.

Author Biography

Paul Sutton is the Navy Interoperability Manager in the Warfare Analysis, Modeling, and Simulation Division, PMW 153, of SPAWAR. He holds a Bachelor of Science degree in General Engineering from the U.S. Naval Academy, and a Master of Science degree in Management from San Diego State University. He has more than thirty years experience in distributed, interactive modeling and simulation, advanced networking engineering, systems engineering, software engineering, and project management. He has previously taught both undergraduate and graduate courses in information systems and management science at U.S. International University, San Diego State University, the University of San Diego, and the University of La Verne.